CSE167: Introduction to Computer Graphics

Lecture #15

Jürgen Schulze
University of California, San Diego
Fall 2008
Announcements

• TA evaluation today

• Final exam tutorial/Q&A on Monday, December 8th 3-5pm, Atkinson Hall room 4004
Final Project

• Problem description is on-line
• Short design description due Monday Dec 1 (email)
• Project due Friday, Dec 5, 1-4pm (demo in lab)
• Can be done in teams of two
• Late submissions deadline Wednesday Dec 10, 2-4pm
StarCAVE Tour

- Location: Atkinson Hall, 1st floor
- Computers: 18 Dell XPS PCs with Quad Core Intel CPUs
- OS: CentOS Linux
- Graphics cards: 2 Nvidia Quadro 5600 per node
- Projectors: 34 JVC HD2k (1920x1080 pixels), ~34 megapixels per eye
- Stereo: passive with circular polarization filters
- 15 screens, ~8 x 4 feet each
- Floor projection
- Optical, wireless tracking system
- Visualization software: COVISE, Electro, CAVElib
- Programming Language: C++

Tour date: Mon 11/24, 5pm
Location: Atkinson Hall, 1st floor
Cubic Bézier patch

- 4x4 mesh of control points

\[ q_0 = \text{Bez}(u, p_0, p_1, p_2, p_3) \]
\[ q_1 = \text{Bez}(u, p_4, p_5, p_6, p_7) \]
\[ q_2 = \text{Bez}(u, p_8, p_9, p_{10}, p_{11}) \]
\[ q_3 = \text{Bez}(u, p_{12}, p_{13}, p_{14}, p_{15}) \]

\[ x(u, v) = \text{Bez}(v, q_0, q_1, q_2, q_3) \]

\[ r_0 = \text{Bez}(v, p_0, p_4, p_8, p_{12}) \]
\[ r_1 = \text{Bez}(v, p_1, p_5, p_9, p_{13}) \]
\[ r_2 = \text{Bez}(v, p_2, p_6, p_{10}, p_{14}) \]
\[ r_3 = \text{Bez}(v, p_3, p_7, p_{11}, p_{15}) \]

\[ x(u, v) = \text{Bez}(u, r_0, r_1, r_2, r_3) \]
Tessellating a Bézier patch

• Uniform tessellation is easiest
  - Evaluate points on a grid of $u$, $v$ coordinates
  - Compute tangents at each point, take cross product to get per-vertex normal
  - Draw triangle strips (several choices of direction)
Piecewise Bézier surface

- $C^0$ continuity: share control points along edges
- $C^1$ continuity: parametric curves that cross each edge need to be $C^1$ continuous

[C^0 continuity](http://www.spiritone.com/~english/cyclopedia/patches.html)  
[C^1 continuity](http://www.spiritone.com/~english/cyclopedia/patches.html)
Questions?
Today

Advanced shader effects

• Environment mapping
• Toon shading
• Shadows
More realistic illumination

- In real world, at each point in scene light arrives from all directions
  - Not just from point light sources
- Environment maps
  - Store “omni-directional” illumination as images
  - Each pixel corresponds to light from a certain direction
Capturing environment maps

- “360 degrees” panoramic image
- Instead of 360 degrees panoramic image, take picture of mirror ball (light probe)

Light probes
[Paul Debevec, http://www.debevec.org/Probes/]
Environment maps as light sources

Simplifying assumption

• Assume light captured by environment map is emitted from infinitely far away

• Environment map consists of directional light sources
  - Value of environment map is defined for each direction, independent of position in scene

• Use single environment map at each point in the scene

• Approximation!
Environment maps as light sources

- How do you compute shading of a diffuse surface using an environment map?
- What is more expensive to compute, shading a diffuse or a specular surface?
**Diffuse Irradiance Environment Map**

- Given scene with $k$ directional lights, light directions $d_1 \ldots d_k$ and intensities $i_1 \ldots i_k$, illuminating a diffuse surface with normal $n$ and color $c$

- Pixel intensity $B$ is computed as:

\[
B = c \sum_{j=1}^{k} \max(0, d_j \cdot n) i_j
\]

- Cost of computing $B$ proportional to number of texels in environment map!
- -> Precomputation of diffuse reflection
- Observations:
  - All surfaces with normal direction $n$ will return the same value for the sum
  - The sum is dependent on just the lights in the scene and the surface normal
- Precompute sum for any normal $n$ and store result in a second environment map, indexed by surface normal
- Second environment map is called *diffuse irradiance environment map*
- Allows to illuminate objects with arbitrarily complex lighting environments with single texture lookup
Diffuse Irradiance Environment Map

- Cube map with diffuse map

- Diffuse shading vs. shading w/diffuse map

Environment maps applications

- Use environment map as “light source”

Global illumination
[Sloan et al.]

Reflection mapping
Cube environment maps

- Store incident light on six faces of a cube instead of on sphere
Cube vs. Spherical Maps

- Advantages of cube maps:
  - More even texel sample density causes less distortion, allowing for lower resolution maps
  - Easier to dynamically generate cube maps for real-time simulated reflections
Cube environment maps

Cube map look-up

- Given direction \((x,y,z)\)
- Largest coordinate component determines cube map face
- Dividing by magnitude of largest component yields coordinates within face
- In GLSL
  - Use \((x,y,z)\) direction as texture coordinates to \texttt{_samplerCube}
Reflection mapping

- Simulate mirror reflection
- Compute reflection vector at each pixel
- Use reflection vector to look up cube map
- Rendering cube map itself is optional
Reflection mapping in GLSL

Application setup

• Load, bind a cube environment map

  glBindTexture(GL_TEXTURE_CUBE_MAP, ...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X,...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X,...);
  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y,...);

  ...

  glEnable(GL_TEXTURE_CUBE_MAP);
Reflection mapping in GLSL

**Vertex shader**

- Compute viewing direction
- Reflection direction
  - Use `reflect` function
- Pass reflection direction to fragment shader

**Fragment shader**

- Look-up cube map using interpolated reflection direction

```glsl
varying float3 refl;
uniform samplerCube envMap;
textureCube(envMap, refl);
```
Questions?
Today

Advanced shader effects

- Environment mapping
- Toon shading
- Shadows
Toon shading

- A.k.a. cel shading
- Simple cartoon style shader
- Emphasize silhouettes
- Discrete steps for diffuse shading, highlights
- Non-photorealistic rendering method (NPR)

Off-line toon shader

GLSL toon shader
Toon Shading Demo

http://www.bonzaisoftware.com/npr.html
Toon shading

- Silhouette edge detection
  - Compute dot product of viewing direction \( \mathbf{v} \) and normal \( \mathbf{n} \)
    
    \[
    \text{edge} = \max(0, \mathbf{n} \cdot \mathbf{v})
    \]

  - Use 1D texture to define edge ramp
    uniform sample1D edgeramp; e=texture1D(edgeramp,edge);
Toon shading

- Compute diffuse and specular shading
  
  \[
  \text{diffuse} = \mathbf{n} \cdot \mathbf{L} \quad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s
  \]

- Use 1D textures `diffuseramp`, `specularramp` to map diffuse and specular shading to colors

- Final color
  
  ```
  uniform sampler1D diffuseramp;
  uniform sampler1D specularramp;
  c = e \times (\text{texture1D(diffuse,diffuseramp)} + \text{texture1D(specular,specularramp)});
  ```
More shaders

- OpenGL shading language book
- NVidia shader library
  - Most shaders are in HLSL (DirectX’s shader
  - NVidia Cg toolkit
  - Current version: Cg 2.1
  - Predecessor of GLSL
  - Lots of example shaders
Questions?
Today

Advanced shader effects

• Environment mapping
• Toon shading
• Shadows
Why are shadows important?

- Give additional cues on scene lighting
Why are shadows important?

- Contact points
- Depth cues
Why are shadows important?

- Realism

Without self-shadowing

With self-shadowing
Terminology

- **Umbra**: fully shadowed region
- **Penumbra**: partially shadowed region
Hard and soft shadows

- Point and directional lights lead to hard shadows, no penumbra
- Area light sources lead to soft shadows, with penumbra
Hard and soft shadows

Hard shadow from point light source

Soft shadow from area light source
Shadows for interactive rendering

• Focus on hard shadows
  - Soft shadows often too hard to compute in interactive graphics

• Two main techniques
  - Shadow mapping
  - Shadow volumes

• Many variations, subtleties

• Active research area
Shadow mapping

Main idea

• Scene point is lit by light source if it is visible from light source

• Determine visibility from light source by placing camera at light source position and rendering scene
Two pass algorithm

First pass

- Render scene by placing camera at light source position
- Store depth image \textit{(shadow map)}

Depth image seen from light source
Two pass algorithm

Second pass

- Render scene from camera position
- At each pixel, compare distance to light source with value in shadow map
  - If distance is larger, we are in shadow
  - If distance is smaller or equal, pixel is lit

Final image with shadows
Questions?
Next time

- Shadow mapping
- Shadow volumes